CPN Tools as a Supplement to UML for Validation of Software Requirements

Vinai George Biju¹ and Santanu Kumar Rath²
¹²Department of Computer Science, NIT Rourkela, Rourkela-769008
Email: ¹grgbiju9@gmail.com, ²rath.santanu@gmail.com

ABSTRACT
In this paper an attempt has been made to describe the dynamic behavior of software development process, particularly during requirements analysis phase using Colored Petri Net (CPN) concepts. It is intended to map system specifications using Unified Modeling Language (UML) diagrams to CPN notations in order to achieve significant advantage. Even though UML is accepted universally by the software industry as a modeling language, it requires strong improvements in order to model the dynamic behavior. One of the shortcomings of UML is the amount of its emphasis on developing executable models. Without executable models, it is quite difficult to investigate behavioral consequences of various design proposals, prior to implementation. None of the behavioral diagrams of UML, in particular the UML State Chart diagram, have a well defined formal semantics. CPNs have formal execution semantics in terms of enabling and occurrence rules and consequently, it indeed offers a true executable model. The main contribution of this paper is to demonstrate that CPN is a powerful supplement to UML by pinpointing a number of specific design issues that can be addressed properly in CPN which is not present in UML. The design issues are further supported by user controlled view of system simulation results.

KEYWORDS
Colored Petri net (CPN), Unified Modeling Language (UML), Use Case Diagram (UCD), Class Diagram (CD).

1. INTRODUCTION
Software development process includes a series of stages such as requirement analysis, design, implementation, testing, reliability assessment, maintenance, and software evolution. One of the most important tasks in software development is to extract the requirements relevant to the process [1]. Incomplete, ambiguous, or even contradictory requirements are recognized by skilled and experienced software engineers at this requirements analysis phase. In this phase the use of right kind of models may help to demonstrate the incorrectness in the requirements, which are very often ambiguous in nature. Hence the identification of incorrectness in requirements helps to proceed towards design phase in a very accurate manner. Validation ensures that the software process remains within established parameters under anticipated conditions. It also includes investigating deviations from established parameters, taking corrective action, evaluating changes in process, procedures, product, equipment, personnel, environment, etc. to determine the effect of change.

The requirements finally agreed upon by the user and the analyst, are prone to exhibit varieties of dynamic behaviors. Dynamic behaviors need to be described by object’s state changes and communication between objects. While identifying the system specification in the form of descriptions, a software analyst finds it difficult to infer the logics and behavior of the system because the description gets vague at times. To overcome this problem, Unified Modeling Language (UML) [2, 3] is used to provide a graphical notation for representing system’s behavior. However, UML model does not provide formal approach of specification and has a weak support for the validation. There are other ways of formally writing the system specification that can be verified using formal techniques and tools. Colored Petri Nets (CPN) is one of such techniques that can be used to model the system’s behavior and check its correctness by analyzing behavioral properties [4]. Thus, before moving towards design phase, the software architect needs to create requirement specification in two formats. One format is to communicate with customers using UML and other format is for analysis and verification purpose using CPN tool [5].

2. RELATIVE WORKS AND COMPARISONS
In some framework, state chart diagrams and collaboration diagrams are also being adopted as the primary notion to model behavior. The dynamic aspect of the system is captured in the state chart diagram as well as the sequence diagram through objects and sequence of messages passed between them [6, 7]. An object Petri Net model of the system has been proposed from the state chart model and collaboration model of the UML [8, 9]. States in the state chart model is mapped to Petri net places and state transitions are mapped to Petri net transitions. In [10] UML sequence diagrams have been transformed to CPN models to verify the correctness of scenarios. Many techniques have been proposed to define mapping from the front end informal notations to formal models. In this paper, the UML model is restricted to the use case model and class model and it is being verified as well as validated by the dynamic behaviors with the corresponding mapped CPN model.

3. ANALYZING THE REQUIREMENTS USING UML
During requirements analysis phase, analyst tries to freeze a set of requirements using a set of use cases, each one having a set of major functionalities. The functionalities later on help to identify the set of stages that a software process is supposed to pass through and the activities involved at every stage.
In the present day, the theoreticians and practitioners consider UML as the de-facto standard modeling language [2, 3]. It is a standardized general-purpose modeling language in the field of software engineering. UML is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software intensive system under development. The UML based system model is represented by means of multiple views such as functional, static and dynamic views. Later UML model is analyzed, especially its dynamic behavior in correlation to properties such as functional correctness and concurrency.

A. Modeling the Functional Requirements

The use case model which describes the sequence of interactions between the actors and system, defines the functional requirements. An actor is a user of the system in a particular role. A use case is a task which an actor needs to perform with the help of system. Associations between actors and use cases are indicated in use case diagrams by solid lines. An association exists whenever an actor is involved with an interaction described by a use case. [3].

Figure 1 depicts the use case model for an example i.e., hospital management system. In this example we have considered the following scenario: it has got several use cases i.e., New patients use case, Doctor Appointments use case and Test/Operation use case etc. The hospital management system has to frequently update their information regarding their patient information, admission discharge reports, ward status as well as details regarding the doctors and staff availability. These set of services are confined to Ward wise bed status use case, Admission discharge reports use case, Patient info use case, Add doctor/staff use case etc. A new patient to the hospital has to get himself registered by the receptionist. Appointments to meet doctor have to be taken and then services like Tests or Operations are received by the patient. Patient will have to clear all the charges which add income to the hospital.

B. Modeling the static architecture

The static model in UML depicts the structural aspects of the system, defines the classes in the system, their attributes and the relationships between classes [2, 3].

In Figure 2, the static structure of the system is documented by class diagram which is a major part of the work of designing an object oriented system. Here classes together with attributes, operations and their association between them are identified for the scenario for a new patient and the services undertaken by the hospital. This module is supposed to work in such a way that, when the patient-ID is generated, he receives the time & date for Appointment from the receptionist and accordingly visits the doctor and undergoes various tests and operations after diagnosis. The patient pays the charges for appointment, tests and operations which add income to the hospital.

4. THE ROLE OF CPN MODELS

CPN [5, 11, 12] is a graphical modeling language suited for modeling concurrency, synchronization, and message passing in different application systems. The CPN modeling language is supported by CPN Tools. [13]. The“CPN Tools” is a tool package that supports the use of CPN concept which is a formal
method that can be used for verification and validation of software systems. UML diagrams model the behavior in a general aspect and have a number of limitations. There are many substantial technical reasons to prefer CPN over UML for e.g. UML state chart diagrams lack well-defined execution semantics, do not support modeling of multiple instances of classes, and do not scale well to large systems etc. The approach of using CPN addresses scalability of model transformation. A large scale UML based system model that is a very complex, flat structure is transformed into a hierarchical CPN model (by use of substitution transitions) which reduces the complexity of initial design [9, 14, 15]. Focusing on use cases may encourage developers to lose sight of the architecture of the system and of the static object structure. Requirements specification by use cases may encourage developers to think too operationally: users are likely to describe use case as a very concrete sequence of interactions with the system which is one way, not the only way, achieving their real goal and as a result there is a danger of missing requirements if much emphasis is put on the suggested process of finding the actors and then finding the use cases that each actor needs. To avoid such a situation it is proposed to develop use case model for each scenario and then subsequently its conceptual model with CPN.

This paper highlights on the aspect of developing Colored Petri Nets (CPN) models to validate the requirements of software system and evaluate systems modeled using UML. The UML based model which is confined to the use case model and class model in terms of functional and static view of the system is mapped to a CPN model. The CPN model as shown in the Figure 3 has a hierarchical structure where an activity that transforms data values or higher level transitions is decomposed to lower level CPN which include sub transitions. A CPN model consists of data, places, transitions and arcs as shown in Figure 3. Location for holding data is known as a place and an activity that transforms data is called as transition. Places and transitions are connected by a directed arc which specifies the data flow paths. CPN color sets and variables are defined in the global area of the CPN model. Tokens represent the data objects and the Color set defines the token type. Tokens of a particular color are placed in locations called places.

Marking of net is the distribution of tokens in respective places of the net. Derivation of color sets and their variables are defined as shown:

- colset recpAvl = av;
- colset patInfoinput = string;
- colset reportsInput = patInfoinput;
- var pinf = patInfoinput; var re = reports input;

Here in this example, colset "repcAvl" is defined with its token named as "av" to indicate the availability of receptionist. The variables "pinf" and "re" are used to extract tokens from places and to put new token into output place. The place"p1" of Figure 3 is of type "repcAvl" with tokens "av". Color sets can be defined from previously declared color sets like for example the color set "reportsInput" indicating the Admission/Discharge reports for a patient is defined from the color set "patInfoinput".

5. TRANSFORMING UML TO CPN MODEL

This section describes the procedures for transforming the UML based system model into CPN model and illustrates each mapping step of the example i.e., hospital management system. The transformation is carried out in the following order: (1) Mapping use case diagram to CPN model, (2) Mapping of the class diagram to the above developed CPN model.

A. Mapping Use case diagram to CPN model

A use case and an actor are respectively mapped to transitions in the CPN model; for example, the actor i.e., receptionist is mapped to a transition in the CPN model and the same way a New patient use case in the use case model is mapped to a New patient transition in the CPN model. Places are added in between the transitions in the CPN model. The color type of the places will be marked from the attributes of the class diagram or marked in CPN model as per requirements and redefined to class model. The functional requirements described in the use case diagram are designed with the CPN model by the mapping method. The correctness of the initial use cases can be validated after simulating the CPN model [12, 17].

B. Mapping class diagram to CPN model

The transition named New Patients in Figure 3 is a substitution transition. This allows CPN to model a hierarchical design. Now each subsystem will be designed in a separate subpage. Here the New patient’s transition designed from the use case model is further decomposed into sub transitions and places which are mapped from the operations associated with the respective classes in the class diagram. The transition named patient in the Figure 4 is mapped from the patient class of the class diagram in Figure 2. The transitions following the patient transition namely createid(), inpatient(), outpatient() are mapped from the operation described in the Patient class in the class diagram.

The patient class interacts with the Registration class for the patient to get registered to the hospital. The validity of the registration is found out when the patient gets registered. This is indicated by the transitions register() and validity() of CPN model in Figure 4 which are mapped from the operations defined in the Registration class in Figure 2.

Similarly modules are designed in CPN as separate subpages decomposing the Doctor Appointments transition as well as Test Operation transition which are substitution transitions in Figure 3.

The CPN model for each scenario is modeled so that the interaction of the classes in the class diagram can be visualized by simulating the CPN model. The places indicated in CPN model are defined from the attributes of classes and even used to redefine the respective classes representing the scenarios.

6. SIMULATION RESULTS AND VALIDATION
The UML based system model is analyzed again after the executable CPN models are developed using the CPN tools package. The first form of analysis validates individual scenarios of use cases and class models of the UML model, using simulation techniques provided by the CPN tools. The passing of the tokens to the places, firing of the transitions and deadlocks can be verified by running the simulation by the user which cannot be illustrated in this paper but rather executing in CPN tool. Corresponding to the inputs initiating the use cases from the actors in the use case model, events in the CPN model are generated to trigger a sequence of internal events for each individual scenario. After analyzing the correctness of each scenario of use cases, and class diagrams, next stage of analysis helps to evaluate concurrency properties of the system in terms of state space diagram and state space reports generated by the CPN tools.

The state space generated for the fig 4 is shown below in fig 5. Each node represents a reachable marking, while each arc represents the occurrence of a single binding element leading from the marking of the source node to the marking of the destination node. To improve readability, we have only shown the detailed contents of one of the markings and some of the binding elements.

![Figure 5: Partial state space for CPN model in figure 4](image)

In addition to state space graph as generated in fig4, reachability graphs can be generated with ordinary Petri Nets using pipe25rc5 tool, only a part of whole model is described in Figure 6 and Figure 7. The reachability graph traces all the possible system states that can be reached by a system to check whether the system is free from deadlock or not. A deadlock free system ensures that the system is not suspended indefinitely. The reachability graph for each use case in the hospital management system, (part of which is depicted in the Figure 7) is shown as an example developed from the concept of ordinary Petri net model in Figure 6. It is a directed graph with nodes representing the marking of the net at any point of time and arcs representing the transition which bring changes in the state. It is always beneficial to analyze the functional correctness of the system by the use of reachability graph as it generates every possible sequence of state changes from the initial state to the final state by allowing any possible firing sequence.

![Figure 6: Petri Net model to generate reachability graph using pipe25rc5 tool](image)
CPN Tools as a Supplement to UML for Validation of Software Requirements

Figure 7: Reachability graph generated using pipe25rc5 tool
In addition to state space graph a state space report can also be generated for large state spaces. It provides great deal of information about the properties of the CPN like Home properties, Liveness Properties, Fairness properties. Explaining about these will be beyond the scope of this paper and it is assumed that the reader has prior knowledge of properties of CPN. These properties aim identification of final states and transitions that will not fire from an initial marking. The report generated for the CPN model in fig 4 is shown below

Statistics
----------------------------------------
State Space
Nodes: 105
Arcs: 226
Secs: 0
Status: Full
Scc Graph
Nodes: 105
Arcs: 226
Secs: 0
Home Properties
----------------------------------------
Home Markings
[105]
Liveness Properties
----------------------------------------
Dead Markings
[105]
Dead Transition Instances
None
Live Transition Instances
None

Fairness Properties
----------------------------------------
No infinite occurrence sequences.

7. CONCLUSIONS AND FUTURE SCOPE
This paper outlines an approach for transforming the UML based functional and static models into hierarchical CPN segments to validate the dynamic behavior during requirements analysis of the software development process. The UML based model represented by use case model and class model in terms of the functional and static view of the software system is transformed to CPN model to form executable models so as to redefine the use cases and class diagrams and improve the quality of requirements.

Using the tool i.e., pipe25rc5, reachability graphs are generated which subsumes all the state transitions that can be reached from an initial state by allowing set of firing sequences. Thus a system can be validated by removing any number of invalid states and deadlocks which enables the software analyst to redefine the use cases and conceptual models. Information regarding number of reachable states, any transitions that doesn’t fire and the properties of CPN model can be analyzed from state space report generated using CPN tool.

The state space graph generated by CPN tools demonstrates whether the model is sound and complete. It contains every possible sequence of state changes from initial state to the final state. Every path in state space graph should be consistent with the desired behavior.

The UML model may be developed using a case tool such as IBM Rational Rose and the CPN model is designed using CPN tools. CPN models can strengthen the behavioral modeling and analysis and can be integrated with software development process to reduce the risk of incorrect designs and increase reliability by user controlled view of system simulations.

For further research in the area of systems analysis, this approach of using UML and CPN tools in an iterative manner can be treated as an authentic procedure in order to improve the Design stages in software engineering. Another extension of this work may be thought of developing methodology for transformation of concurrent composite state chart diagrams and exploring techniques for automatic generations of CPN from UML diagrams.

8. REFERENCES

[6]. L. Amorim, P. Maciel, M. Nogueira, R. Barreto, and


Figure 3: CPN model developed from use case diagram

Figure 4: CPN model mapped from the class diagram